МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7381 _____ Лукашев Р.С.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

Задачи.

- Ознакомиться с представлением графических данных;
- Ознакомиться с простейшим способом передачи графических данных;
- Создать модель;
- Настроить параметры обучения;
- Написать функцию, позволяющую загружать изображение пользователем и классифицировать его.

Требования.

1. Найти архитектуру сети, при которой точность классификации будет не менее 95%;
2. Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения;
3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета.

Ход работы.

Данная изначально архитектура сети удовлетворяет условиям задания. Изменения в архитектуре (добавление новых слоев и изменение количества нейронов на слоях) не сильно влияют на точность классификации. Для экономии времени было решено уменьшить количество нейронов на втором слое в два раза (128 нейронов), при этом разница в точности классификации составила примерно 0.5% в среднем.

Исследуем влияние оптимизаторов и скоростей обучения на точность модели. Будем исследовать оптимизаторы RMSprop, Adam, Nadam, Adamax, Adagrad, Adadelta и SGD на стандартных настройках, варьируя скорость обучения. Тренировка проводилась в 5 поколений, размер пакета – 128. Необходимая точность достигалась в основном на 2 – 3 поколении, при этом было решено оставить пять поколений для надежного сравнения моделей. Результаты приведены на рис. 1 – 4.

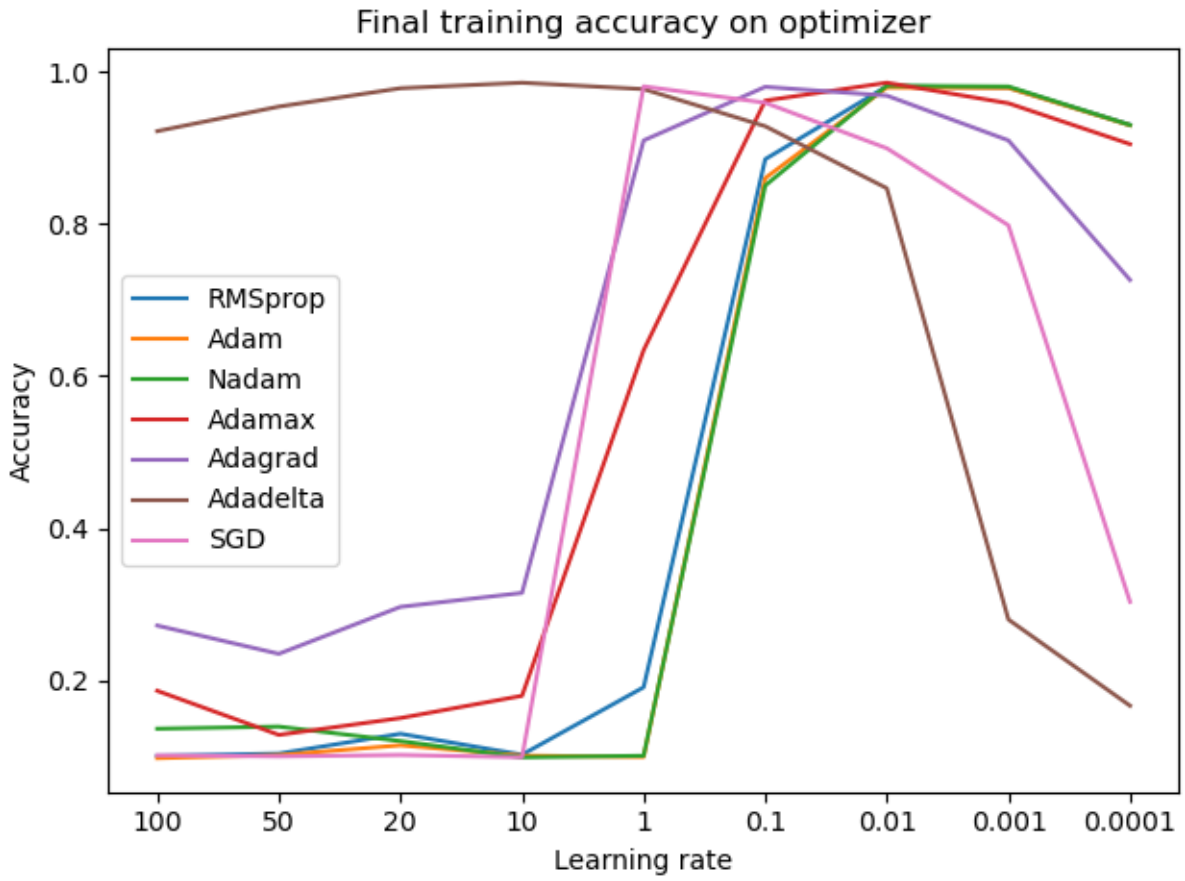

Рис. 1 – График достигнутой в конце обучения точности модели в зависимости от скорости обучения.

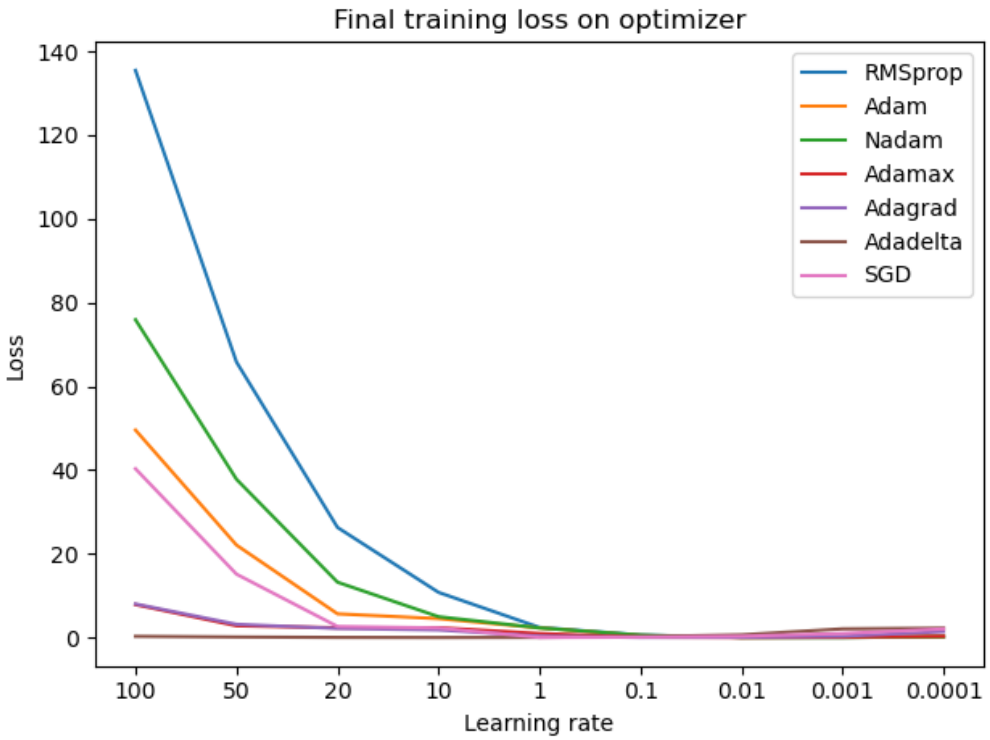

Рис. 2 – График достигнутых в конце обучения потерь модели в зависимости от скорости обучения.

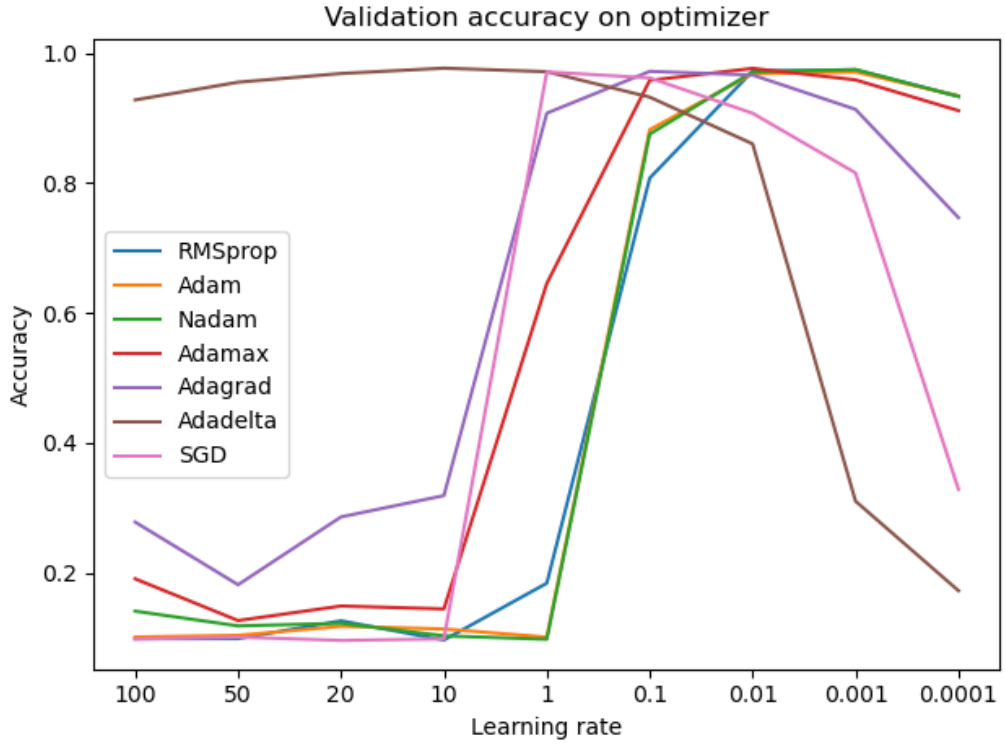

Рис. 3 – График точности модели на тестовых данных в зависимости от скорости обучения.

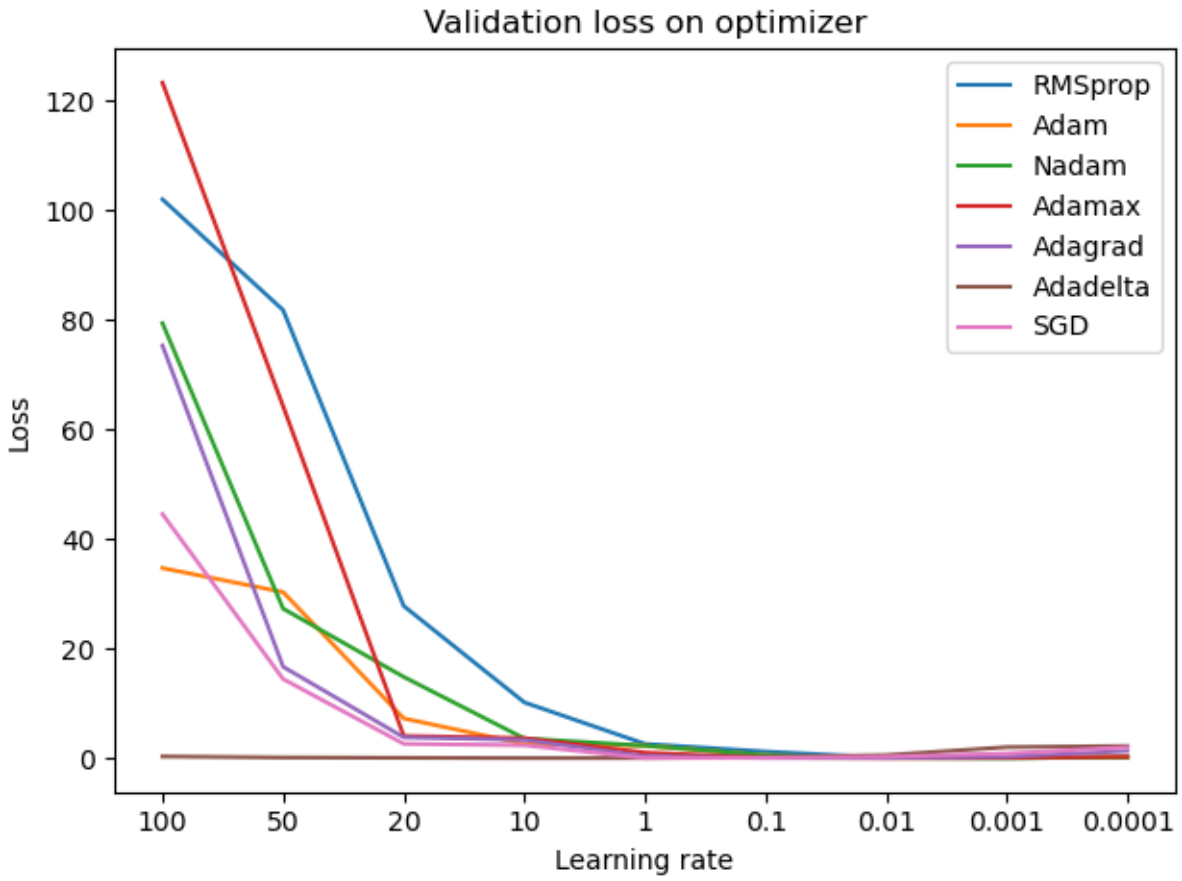

Рис. 4 – График потерь модели на тестовых данных в зависимости от скорости обучения.

Как можно увидеть, при слишком больших значениях скорости обучения модель может оказаться необучаемой, при слишком маленьких, модель обучается дольше. На графиках выделяется оптимизатор Adadelta, который хорошо обучает модель при больших значениях скорости обучения, и хуже при маленьких. Это объясняется тем, что данный оптимизатор сам модифицирует значение скорости обучения в процессе обучения. Параметр отвечает за начальное значение скорости обучения. Таким образом, при больших значениях он может быстро добраться до оптимальных, и обучаться при оптимальных значениях, а при маленьких это происходит гораздо медленнее.

При определенных значениях скорости сходимости все оптимизаторы смогли достаточно хорошо обучиться, однако при параметрах по умолчанию хуже всех себя показывает оптимизатор SGD. Adadelta – немного хуже

остальных (Adam, Nadam, RMSprop, Adamax и Adagrad), которые дали схожие результаты.

Оптимизаторы Adam, Adamax и Nadam используют два параметра – beta_1 и beta_2 – соответственно экспоненциально убывающие коэффициенты обновления смещенной оценки первого и второго моментов. Посмотрим на влияние значений этих коэффициентов на точность обученной модели:

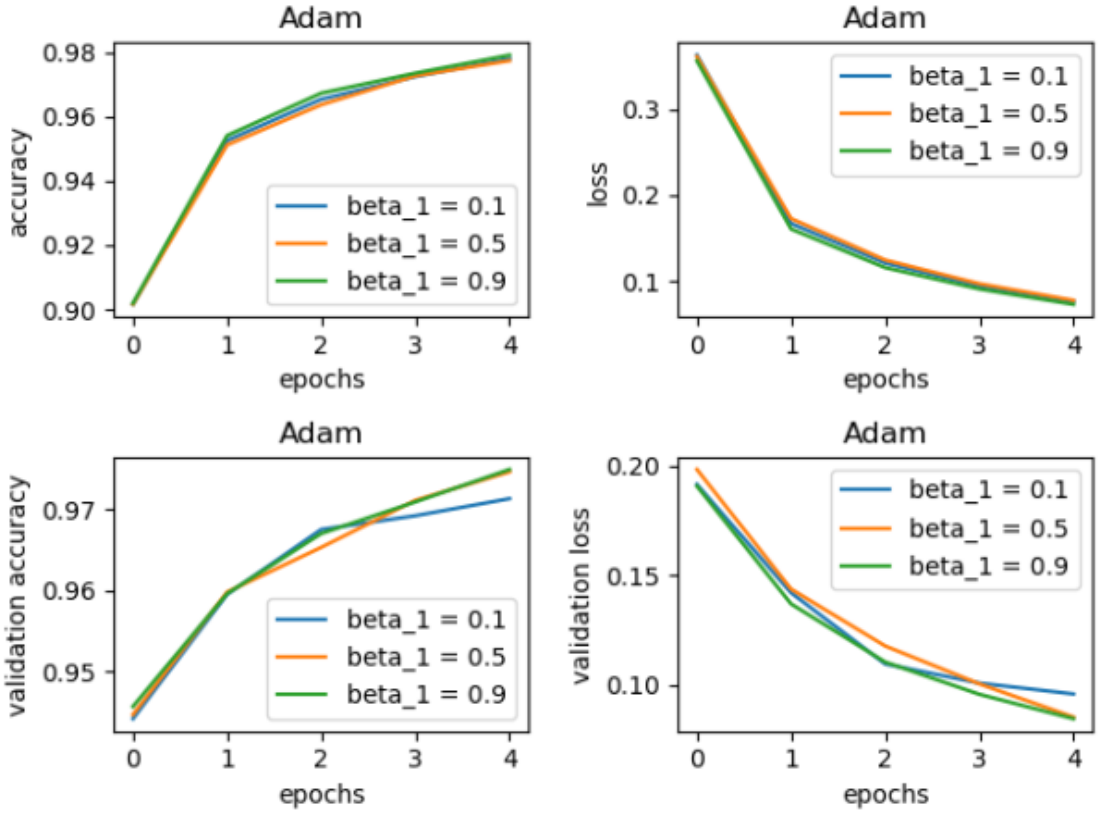

Рис. 5 – Графики ошибки и точности модели в зависимости от значения параметра beta_1.

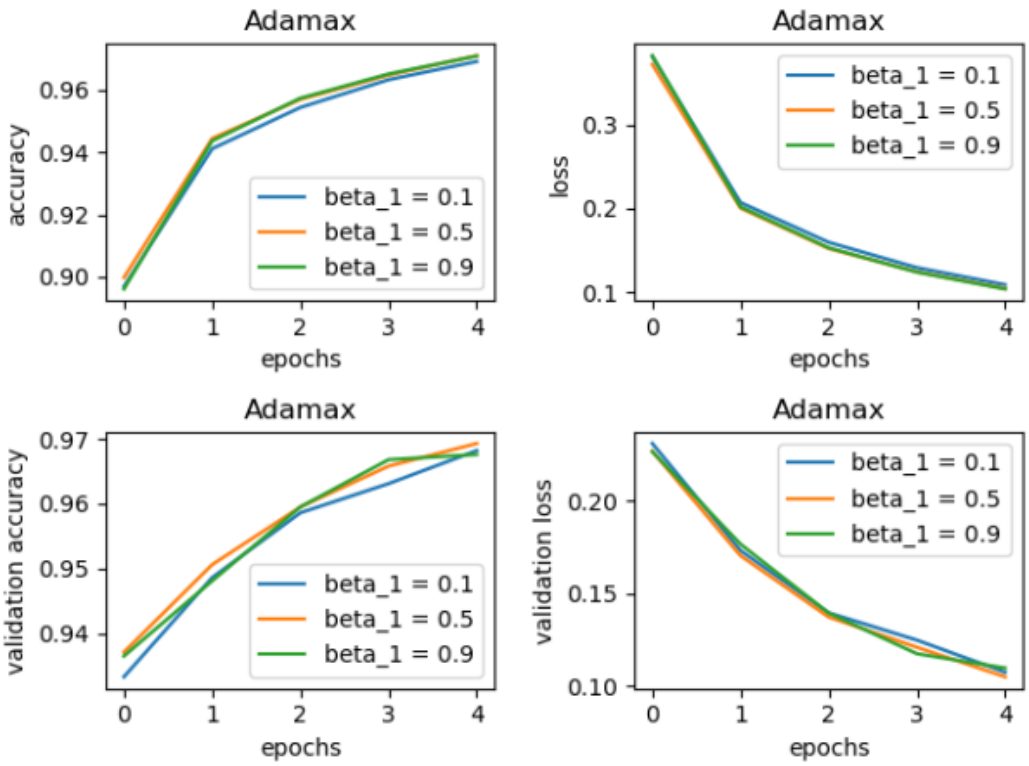

Рис. 6 – Графики ошибки и точности модели в зависимости от значения параметра beta_1.

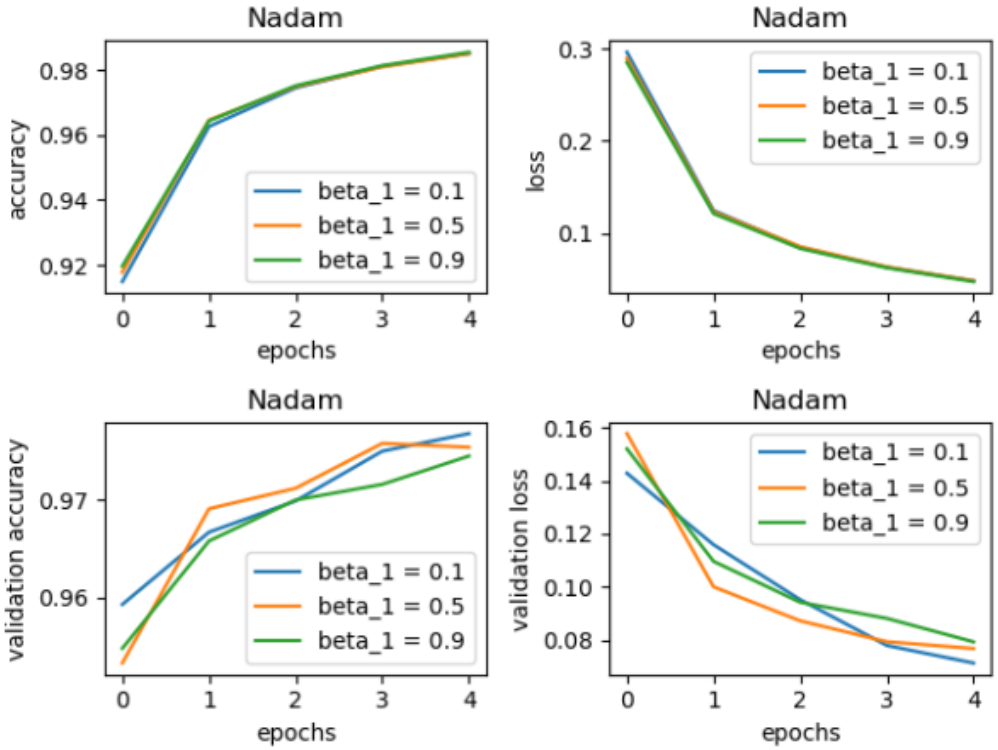

Рис. 7 – Графики ошибки и точности модели в зависимости от значения параметра beta_1.

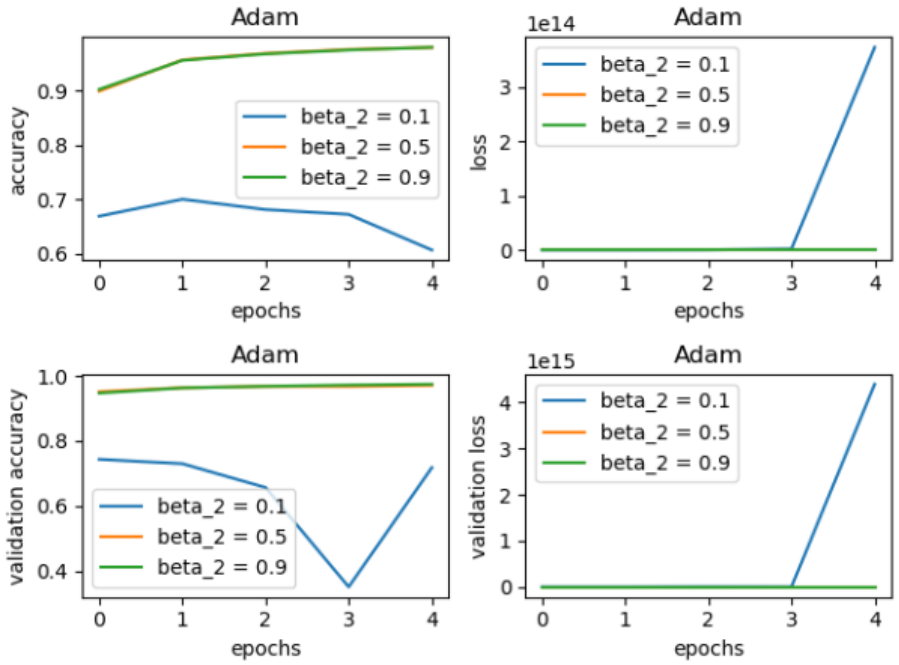

Рис. 8 – Графики ошибки и точности модели в зависимости от значения параметра beta_2.

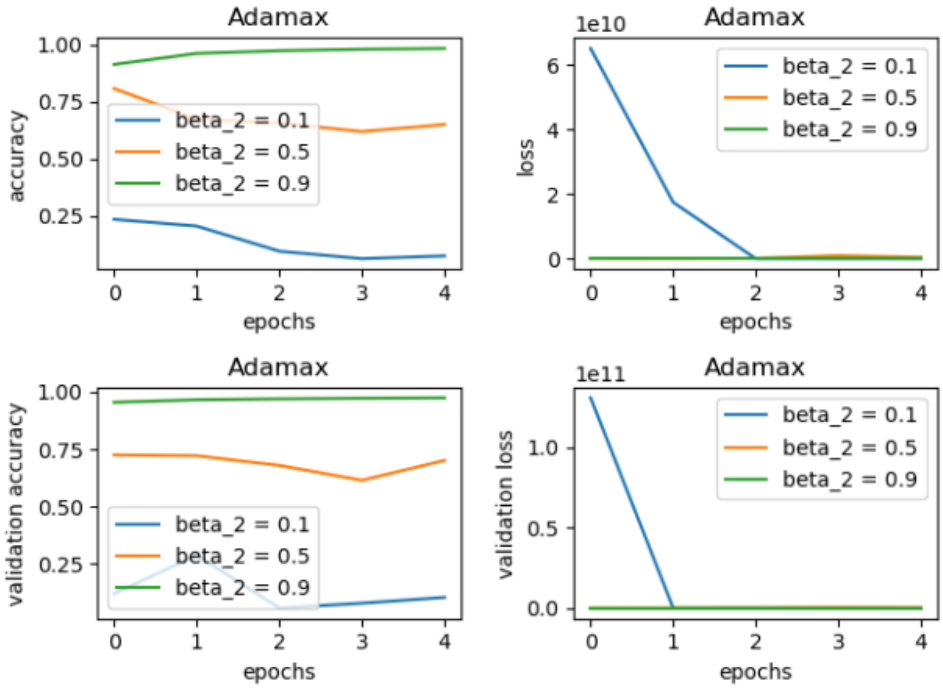

Рис. 9 – Графики ошибки и точности модели в зависимости от значения параметра beta_2.

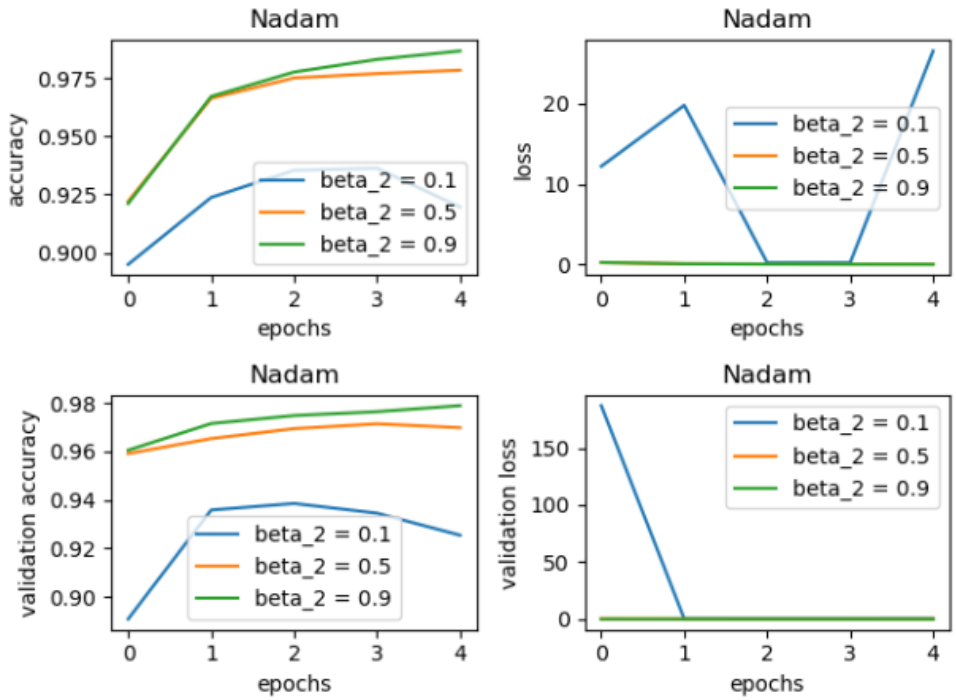

Рис. 10 – Графики ошибки и точности модели в зависимости от значения параметра beta_2.

Как можно заметить, чем меньше значения этих параметров, тем хуже обучается модель. При этом beta_2 значительно сильнее влияет на результаты обучения чем beta_1, и оптимизатор Nadam сравнительно слабо чувствителен к этому параметру.

Оптимизатор Adam имеет два варианта – Adam и AMSGrad. Для переключения между ними используется параметр. На рисунке 11 показаны результаты обучения на обоих вариантах:

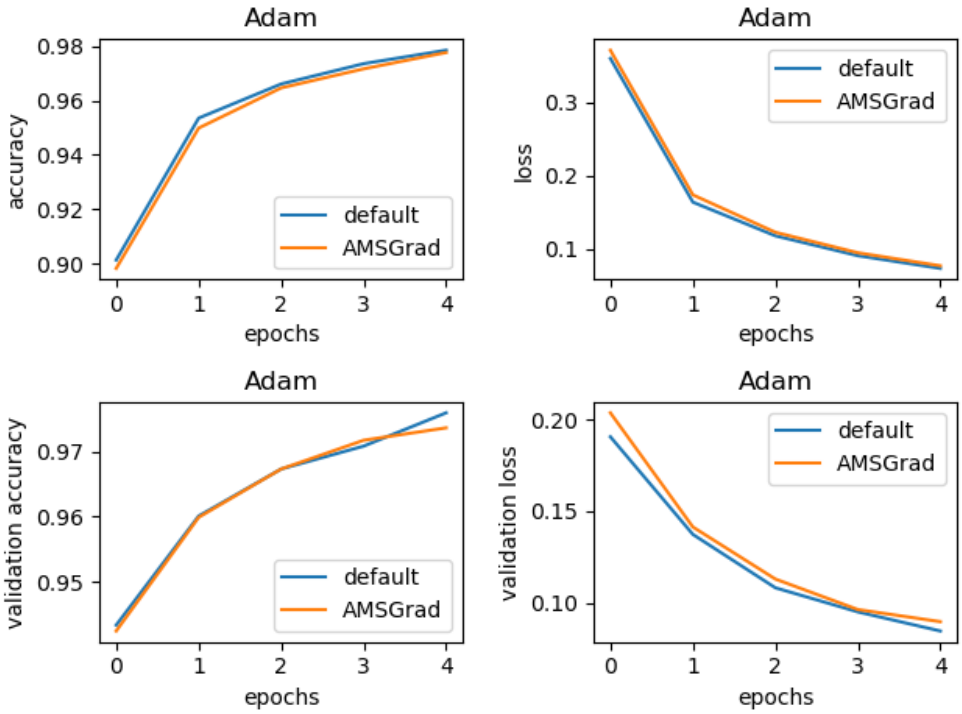

Рис. 10 – Графики ошибки и точности модели двух вариантов оптимизатора Adam.

Как можно увидеть, особой разницы между результатами нет.

Оптимизатор SGD оперирует с параметром momentum (импульс), ускоряющим оптимизатор. Также есть опция использования импульса Нестерова. Результаты обучения при различных импульсах показаны на рис. 11 – 12.

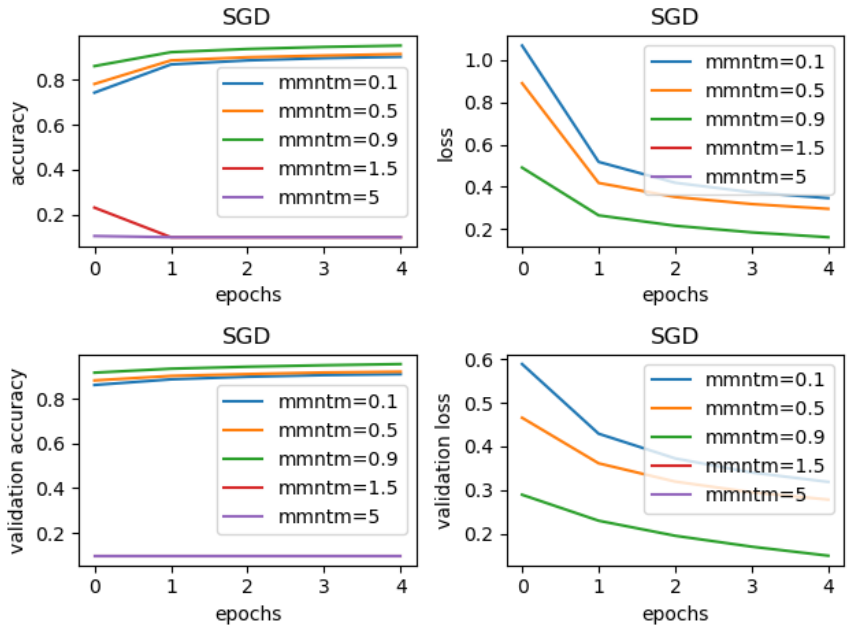

Рис. 11 – Графики ошибки и точности модели в зависимости от момента.

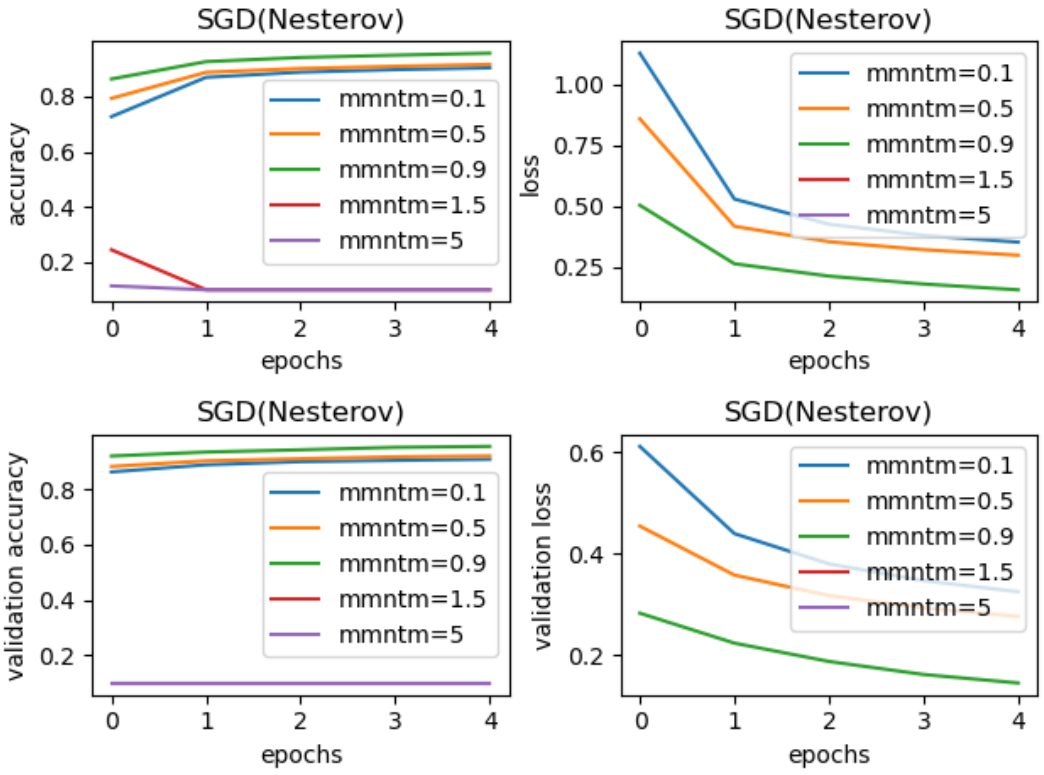

Рис. 12 – Графики ошибки и точности модели в зависимости от момента.

Как можно увидеть, разницы между результатами с использованием стандартного импульса и импульса Нестерова практически нет. При этом значения импульса, близкие к единице проявляют себя лучше всего. Слишком большие значения делают модель необучаемой. Также следует отметить, что при хорошо подобранном значении момента, модель показывает себя лучше остальных (по умолчанию параметр равен 0.0).

Оптимизатор RMSprop работает с параметром rho – коэффициентом убывания градиента. Результаты обучения при различных значениях этого параметра представлены на рисунке 13. Как можно увидеть, маленькие значения rho (меньшие единицы) очень слабо влияют на результат обучения, а слишком большие делают модель необучаемой.

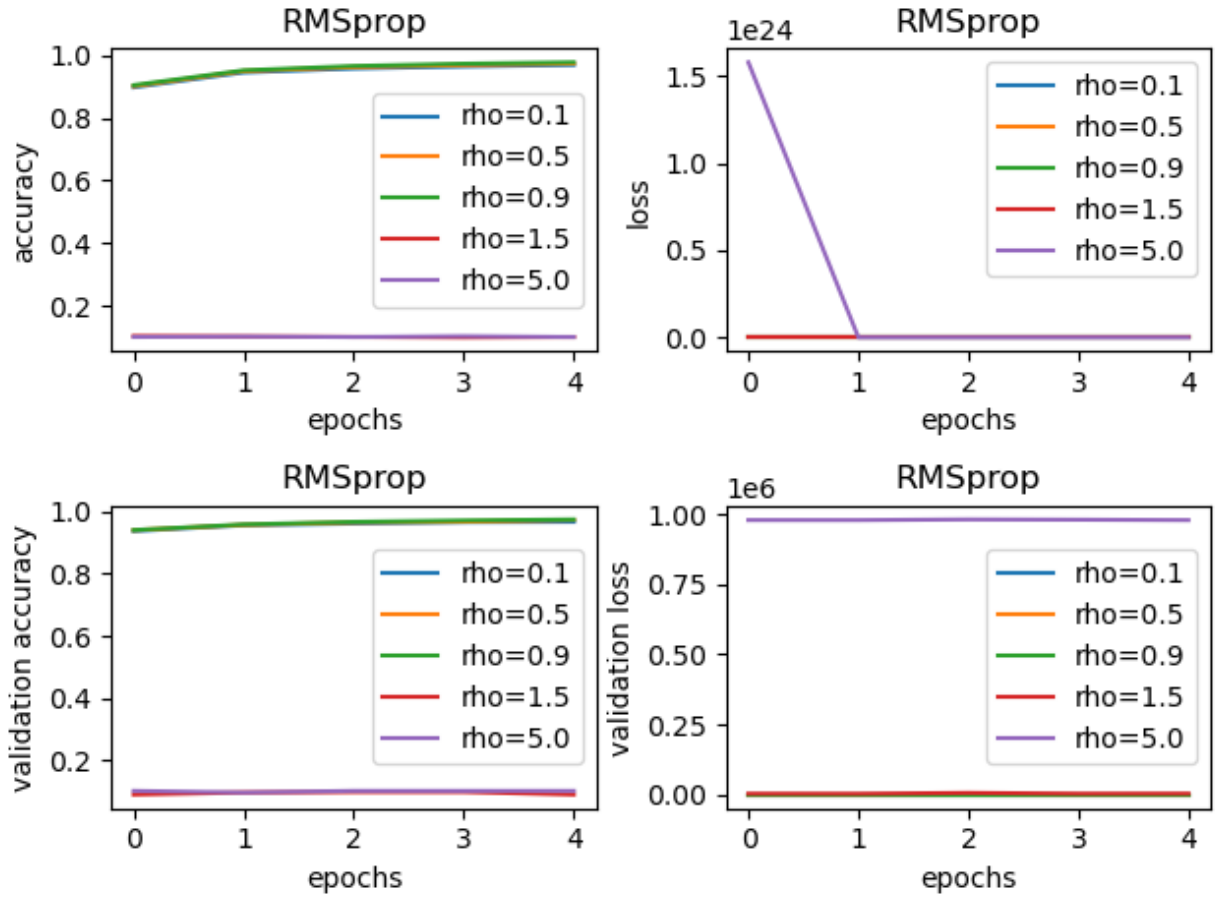

Рис. 13 – Графики ошибки и точности модели в зависимости от коэффициента убывания градиента.

Также все оптимизаторы работают с параметрами clipnorm и clipvalue. При их указании значения градиентов будут ограничиваться нормой clipnorm, и находиться в интервале [clipvalue, -clipvalue]. Они используются для решения проблем взрывающихся и исчезающих градиентов.

Была написана функция, позволяющая загружать и классифицировать пользовательские картинки:

```
def user_image_predict(image_path, model):
    image = np.expand_dims(np.array(Image.open(image_path).convert('L').resize((28, 28))), axis=0)
    image = image / 255
    return model.predict_classes(image)[0]
```

Загрузка и преобразование изображения производится с использованием библиотеки Pillow. Для конвертации изображения в grayscale необходимо использовать метод convert с параметром ‘L’.

Пример работы функции показан на рис. 14.

```
test_acc: 0.9733999967575073 test_loss: 0.09063528722040355
actual - 9, predicted - 9
actual - 5, predicted - 5
actual - 3, predicted - 3
```

Рис. 14. - Демонстрация работы функции классификации пользовательского изображения.

Выводы.

В ходе выполнения данной лабораторной работы были получены навыки по построению сетей для классификации изображений. В рамках исследования была создана и обучена модель, распознающая рукописные символы (цифры от 0 до 9). Также была написана функция, позволяющая классифицировать пользовательские изображения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
from matplotlib import pyplot as plt
from keras.layers import Dense, Flatten
from keras.models import Sequential
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.optimizers import *
from PIL import Image


def build_model(optimizer):
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy']
                  )
    return model



def user_image_predict(image_path, model):
    image = np.expand_dims(np.array(Image.open(image_path).convert('L').resize((28,
28))), axis=0)
    image = image / 255
    return model.predict_classes(image)[0]



def line_plot_against(data, name, title, xlabel, ylabel):
    fig, ax = plt.subplots()
    for d in data:
        plt.plot(d)
    ax.set_xticks(np.arange(9))
    ax.set_xticklabels(['100', '50', '20', '10', '1', '0.1', '0.01', '0.001',
'0.0001'])
    plt.legend(['RMSprop', 'Adam', 'Nadam', 'Adamax', 'Adagrad', 'Adadelta', 'SGD'])
```

```
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    fig.tight_layout()
    plt.savefig(name)
    plt.show()
    plt.clf()


def variable_parameters_test(optimizer_list, names, file_prefix, legend):
    i = 0
    for data in optimizer_list:
        name = ''
        for optimizer in data:
            model = build_model(optimizer)
            H = model.fit(train_images, train_labels,
                          epochs=5, batch_size=128,
                          validation_data=(test_images, test_labels),
                          verbose=0
                          )
            plt.subplot(221)
            plt.xlabel('epochs')
            plt.ylabel('accuracy')
            plt.plot(H.history['accuracy'])
            plt.subplot(222)
            plt.xlabel('epochs')
            plt.ylabel('loss')
            plt.plot(H.history['loss'])
            plt.subplot(223)
            plt.xlabel('epochs')
            plt.ylabel('validation accuracy')
            plt.plot(H.history['val_accuracy'])
            plt.subplot(224)
            plt.xlabel('epochs')
            plt.ylabel('validation loss')
            plt.plot(H.history['val_loss'])
        name = names[i]
```

```
        i += 1
        plt.legend(legend)
        plt.title(name)
        plt.subplot(221)
        plt.legend(legend)
        plt.title(name)
        plt.subplot(222)
        plt.legend(legend)
        plt.title(name)
        plt.subplot(223)
        plt.legend(legend)
        plt.title(name)
        plt.savefig(file_prefix + name + '.png')
        plt.show()
        plt.clf()


(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images, test_images = (train_images / 255.0, test_images / 255.0)
train_labels, test_labels = to_categorical(train_labels), to_categorical(test_labels)

# learning rate comparison
train_acc, test_acc = [[],[],[],[],[],[],[]], [[],[],[],[],[],[],[]]
train_loss, test_loss = [[],[],[],[],[],[],[]], [[],[],[],[],[],[],[]]
i = 0
for optimizer in [RMSprop, Adam, Nadam, Adamax, Adagrad, Adadelta, SGD]:
    for lr in [100, 50, 20, 10, 1, 0.1, 0.01, 0.001, 0.0001]:
        model = build_model(optimizer(lr=lr))
        H = model.fit(train_images, train_labels,
                      epochs=5, batch_size=128,
                      validation_data=(test_images, test_labels),
                      verbose=0
                      )
        train_acc[i].append(H.history['accuracy'][-1])
        test_acc[i].append(H.history['val_accuracy'][-1])
        train_loss[i].append(H.history['loss'][-1])
        test_loss[i].append(H.history['val_loss'][-1])
```

```
    i += 1


line_plot_against(train_acc, 'training_acc.png', 'Final training accuracy on
optimizer', 'Learning rate', 'Accuracy')

line_plot_against(test_acc, 'test_acc.png', 'Validation accuracy on optimizer',
'Learning rate', 'Accuracy')


line_plot_against(train_loss, 'training_loss.png', 'Final training loss on
optimizer', 'Learning rate', 'Loss')

line_plot_against(test_loss, 'test_loss.png', 'Validation loss on optimizer',
'Learning rate', 'Loss')


# other parameters

i = 0

optimizer_list = [[], [], []]

for optimizer in [Adam, Adamax, Nadam]:

    for beta in [0.1, 0.5, 0.9]:

        optimizer_list[i].append(optimizer(beta_1=beta))

    i += 1


variable_parameters_test(optimizer_list, ['Adam', 'Adamax', 'Nadam'], 'beta_1_',

                         ['beta_1 = 0.1', 'beta_1 = 0.5', 'beta_1 = 0.9'])


i = 0

optimizer_list = [[], [], []]

for optimizer in [Adam, Adamax, Nadam]:

    for beta in [0.1, 0.5, 0.9]:

        optimizer_list[i].append(optimizer(beta_2=beta))

    i += 1


variable_parameters_test(optimizer_list, ['Adam', 'Adamax', 'Nadam'], 'beta_2_',

                         ['beta_2 = 0.1', 'beta_2 = 0.5', 'beta_2 = 0.9'])


optimizer_list = [[Adam(amsgrad=False), Adam(amsgrad=True)]]

variable_parameters_test(optimizer_list, ['Adam'], 'AMSGrad_', ['default',
'AMSGrad'])


optimizer_list = [[], []]

for momentum in [0.1, 0.5, 0.9, 1.5, 5]:
```

```
    optimizer_list[0].append(SGD(nesterov=False, momentum=momentum))
    optimizer_list[1].append(SGD(nesterov=True, momentum=momentum))

variable_parameters_test(optimizer_list, ['SGD', 'SGD(Nesterov)'], 'Momentum_',
                         ['mmntm=0.1', 'mmntm=0.5', 'mmntm=0.9', 'mmntm=1.5',
'mmntm=5'])

optimizer_list = [[], []]
for rho in [0.1, 0.5, 0.9, 1.5, 5]:
    optimizer_list[0].append(RMSprop(rho=rho))
    optimizer_list[1].append(Adadelta(rho=rho))

variable_parameters_test(optimizer_list, ['RMSprop', 'Adadelta'], 'Rho_',
                         ['rho=0.1', 'rho=0.5', 'rho=0.9', 'rho=1.5', 'rho=5.0'])

# user data demonstration
model = build_model(Adam())
H = model.fit(train_images, train_labels,
              epochs=5, batch_size=128,
              validation_data=(test_images, test_labels),
              verbose=0
              )

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc, 'test_loss:', test_loss)

print("actual - 9, predicted -", user_image_predict('9.png', model))
print("actual - 5, predicted -", user_image_predict('5.png', model))
print("actual - 3, predicted -", user_image_predict('3.png', model))
```